Asynchronous SAN Switching under Multicast Traffic

Asynchronous SAN Switching under Multicast Traffic

Andrea Bianco *, Luca Giraudo *, Alessandra Scicchitano †
* Dip. di Elettronica, Politecnico di Torino, Italy, Email: {name.surname}@polito.it
† IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland, Email: {als}@zurich.ibm.com

Abstract— **Multicast traffic in Storage Area Networks (SANs) enables applications such as disaster recovery, remote data replication and distributed multimedia systems, in which a server access concurrently multiple storage devices or, conversely, multiple servers access data on a single device. Thus, an asynchronous loss-less switching architecture devised for SANs is described, and its performance under multicast traffic is studied. Simulations are used to analyze switch performance under various traffic patterns and schedulers. Although most of the simulations refer to a specific switch architecture, performance results highlight interesting general trends in flow controlled asynchronous architectures. These architectures could be used effectively also in a more traditional data switching and routing scenario. In this case, multicast support becomes essential to support multimedia QoS aware applications and protocols heavily relying on the broadcast property of LANs.**

I. Introduction

High-speed packet-switched networks, named Storage Area Networks (SANs), are replacing direct connections between servers and storage resources. Indeed, SANs provide more flexibility, overcome the performance, scalability, reliability and management problems of the traditional Directly Attached Storage (DAS) paradigm, and enable consolidation and virtualization of storage resources. To ensure reliability, loss-free operation is envisioned. Both buffer-to-buffer and end-to-end flow-control mechanisms are proposed in Fibre Channel standards [1] to control the rate at which frames are received from upstream nodes to avoid frame losses and reordering.

Multicast support in SANs enables critical applications such as disaster recovery, in which a server stores multiple copies of the same data at geographically distant sites (similar to RAID-1 mode), and distributed multimedia systems [2], in which multiple servers access data (typically video streams) stored in a central repository and deliver it to their local pool of users [3].

In this paper we present a switch architecture designed for SANs, and study its performance under multicast traffic. Multicast packets are characterized by their fanout set, i.e., by the set of output ports (destinations) to which they are directed. The packet fanout is defined as the number of different destinations of a multicast packet, i.e., the cardinality of the fanout set. All the packets arriving to the same input and with the same fanout set identify a multicast flow. Unicast traffic is not given special attention, i.e., it is considered as a particular case of multicast traffic with fanout 1.

The presented architecture was previously introduced in [4], [5]: it employs backpressure to achieve lossless behavior. The switch is fully *asynchronous*, since asynchronous behavior provides significant advantages in terms of scalability, cost and simplicity [6]. Indeed, traditional input-queued switches operate in a *synchronous* fashion: time is divided in intervals of equal size called *time-slots* and modules (line-cards, fabric, scheduler) have a common time reference. Variable-size packets are segmented into fixed-size data units called *cells*, transferred through the switching fabric within a time-slot and reassembled at the output line-cards. In an asynchronous switch, on the contrary, line-cards and the switching fabric run on independent clock domains. As such, global clock distribution is not needed, thus avoiding a very complex task especially when the system is distributed over multiple racks. Furthermore, no synchronized transmission through the switching fabric is required and variable-length packets can be supported natively, without the need for segmentation and reassembly buffers. Finally, fabric arbitration is simplified because output contentions can be solved independently, without employing complex centralized scheduling algorithms.

II. System Architecture

The switch is composed by a buffered switching fabric, and a given number of line-cards, comprising input and output buffers, as shown in Fig. 1. Every line-card is composed by an input port and an output port (port multiplexing is not considered for simplicity). Line-cards receive packets and store them in input buffers. The switching fabric transfers (multicast) packets from the line-card input buffer to the line-cards hosting the destination output ports, exploiting its intrinsic multicast capability. Switching fabric I/O links are not oversubscribed nor constitute a bottleneck. Backpressure control signals regulate buffer access to avoid data loss.

A. Line-cards

Line-cards contain two separate buffering stages for multicast packets entering and exiting the switch, named respectively *In-module* and *Out-module*. The In-module memory is organized as a single FIFO queue. Each position in the queue is dimensioned for a maximum transfer unit (MTU). If a smaller packet is enqueued, the residual part of that memory portion remains unusable. This choice potentially results in inefficient use of the In-module memory when dealing with small-size packets, but the buffer management policy can be implemented using simply one counter. Normally this is not a major issue, because line-cards can host a moderately large amounts of

978-1-4244-1982-1/08/$25.00 ©2008 IEEE

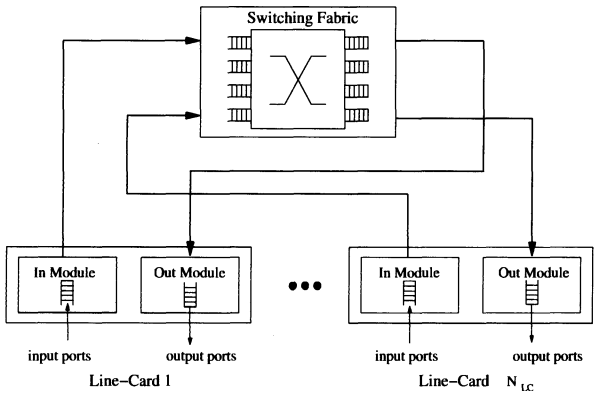

Fig. 1. Switch architecture: components and communications channels

memory. The Out-module stores multicast packets received from the switching fabric in a buffer organized as a single FIFO queue. This very fast memory is accessed "per-byte", hence the number of available positions depends on the size of enqueued packets.

B. Switching fabric

The switching fabric consists of a crossbar with no internal buffers in crosspoints, but with a small single on-chip high-speed FIFO queue at each input and output. Buffers in the switching fabric are needed due to the fully asynchronous switch behavior. The crossbar may have a moderate internal speed-up K to mitigate the effect of Head-of-the-line (HOL) blocking of FIFO queues; however, in the simulation analysis, we consider $K = 1$. Fabric output queues are larger than fabric input queues to sustain temporary overload conditions. Both input and output queues are accessed per-byte to maximize space efficiency.

Each fabric output has a *fabric scheduler* that controls access from fabric inputs. When an input wants to be connected to an output, it sends a *request* to the corresponding output fabric scheduler. The output scheduler sends *grants* to inputs. In case multiple inputs request the same output, the output scheduler solves the contention according to a round-robin (or random) policy.

The crossbar has an internal multicasting capability: it can replicate a packet to multiple outputs at the same time with no extra cost. The asynchronous behavior poses different challenges with respect to synchronous slotted switching when dealing with multicast traffic. In synchronous switching, decision are taken synchronously by all output schedulers at time slot boundaries. As such, no output remains unnecessarily idle if enough traffic is available at inputs. When considering asynchronous behavior, output schedulers make independent decisions at different times. However, some sort of grant synchronization at inputs can be useful, to avoid sending multicast packets only according to a multi-copy scheme, i.e.. as independent unicast packets. Indeed, the multi-copy approach is known to be an inefficient scheduling technique. For example, when considering as an admissible traffic pattern a single broadcast flow in overload at a given input, the multi-copy approach gives a maximum throughput of $1/N$. On the other hand, waiting to gain access to all the intended outputs before transmitting a packet can be counterproductive, because it forces outputs that have already granted access to stay idle while the other outputs become free.

To exploit the benefits of crossbar replication without compromising efficient usage of output ports, the formerly proposed scheduling scheme [5] was based on three phases:

- *Waiting phase*: every input sends a request for every output in the fanout set of the HoL packet stored in the fabric input buffer. Each input collects the grants received from the outputs until a timeout T expires;
- *Multicast transmission phase*: After timeout expiration, each input sends the multicast packet to every output that granted its request, using the internal multicast capability of the crossbar;
- *Unicast transmission phase*: If the input has not received the grants from all outputs during the waiting phase, it sends an individual copy of the packet to the remaining destinations as soon as each output grants the request.

The timeout is used to obtain a "grant synchronization" effect at inputs: inputs waiting for grants from outputs belonging to the multicast packet fanout set may better exploit the fabric multicast capability if more outputs grant the request during timeout expiration. Indeed, the larger the number of received grants, the smaller the number of transmissions required to transfer a multicast cell to the outputs in the fanout set. On the other hand, while waiting for timeout expiration, no transmission occur: thus, the timeout value must be carefully set to balance these two effects on performance. In other words, strictly enforcing a no-fanout splitting policy, i.e., a multicast packet is transferred only once, when all the outputs in the fanout set are available, may induce performance degradation due to blocking. On the other hand, splitting the multicast packet in too many transmissions, when using a fanout splitting policy, increases too much the load on the switching fabric, thus, reducing performance.

The modification we propose to enhance switch performance is to substitute the unicast transmission phase with a number of multicast transmission phases until the multicast packet is fully transmitted. This modification requires a minor complexity increase, and provides significant benefits. Note that no timeout expiration is necessary after the first multicast transmission phase, since inputs aggregate grants received from outputs during the packet transmission time. Furthermore, the adoption of the multicast transmission phase only, permits, as shown later, to avoid the use of the timeout to synchronize grants, a parameter whose value should be set properly to obtain good performance.

This scheduler is named round robin scheduler in the remainder of the paper, since multiple requests received by an output scheduler while the output is engaged in a packet transmission are served according to a round robin order. It is well known that round-robin (or random) schedulers do not perform particularly well, since no information on the relative

importance or urgency of packets is used when selecting the input to which the request is granted. A better solution can be to use some "weighted" metric when sending requests from inputs and when selecting the input to which to issue the grant at outputs. Examples of weighted schedulers defined for synchronous switches are LQF (Longest Queue First) [8] for unicast traffic, WBA (Weight Based Arbiter) [9] and GS (Greedy Scheduler) [10] for multicast traffic. One natural choice of weight is the queue length (LQF), since the longer the queue the higher the input load; the scheduler tries to favor highly-loaded inputs to improve performance. However, since the maximum queue length is finite, this weight is significant only when losses are not experienced, i.e., in low-medium loads. On the contrary, all the queues experiencing losses have constant queue length, independently from their congestion level and the queue length metric would not help in this scenario. When presenting WBA, several metrics were proposed and compared. Given the similar performance provided by the various metrics, we choose the weight equal to the number of inputs minus the packet fanout plus the packet age (measured as the difference between the current time minus the time at which the packet entered the queue). Indeed, packet age, although being a complex metric to be managed, permits to discriminate packets experiencing long starvation periods. Moreover, taking into account also the packet fanout permits to favor multicast packets with large fanout sets, packets known to be more difficult to schedule. Finally, we also considered the GS weight, the product of the queue length by the actual fanout size of the packet at the head of the queue. In contrast with the WBA metric, the GS metric does not require any packet delay computation.

However, two major differences can be highlighted in asynchronous switching with respect to the more traditional synchronous scenario. First, in synchronous switches, all outputs receive weight information at the same time, at slot boundaries, from all inputs. As such, coordination among output selectors can be envisioned to optimize packet selection. Indeed, since all outputs make an input selection in each time slot, it is more likely that several outputs select the same input request if the associated weight is much larger than other weights. Second, due to the packet segmentation process at inputs and to the cell-based packet transfer in the switching fabric, several requests with different weights are sent for a given packet in consecutive time slots, until the multicast packet is fully extracted from the input queue. None of these two properties hold in asynchronous switches, since each output selects a new request independently, when a packet transmission ends, and the request is issued only once, when the packet reaches the head of the corresponding FIFO queue. As such, weighted metrics could be less effective than in the synchronous case.

Finally, also multicast schedulers like MRR cannot be easily used in an asynchronous scenario. Indeed, MRR is based on the idea of keeping, in all outputs, a common pointer (a modulo N counter) to inputs. The pointer is used to preferentially grant, according to a round-robin scheme, requests incoming from inputs in a given time slot. This common reference clearly favors the possibility of selecting the same input at many outputs, thus preserving as much as possible the no-fanout splitting property of the scheduler. This scheduler cannot be used in asynchronous switches, since no coordination can be easily enforced among outputs.

To summarize, besides the round-robin scheduler, three weighted schedulers running on three different metrics are also studied:

- LQF metric: each request contains the length of the FIFO queue at the corresponding input;
- WBA metric: the weight is equal to the number of inputs minus the cell fanout plus the packet age;
- GS metric: the weight is the product of the queue length by the actual fanout size of the cell at the head of the queue.

C. Control mechanisms for lossless delivery

To support lossless delivery, the switch adopts an internal backpressure mechanism that regulates access to buffers to prevent overflow. When the buffer occupancy overcomes a *high threshold*, a backpressure signal is activated to block packet transmissions from upstream buffering stages. When the buffer occupancy becomes smaller than *a low threshold*, the backpressure signal is deactivated and transmission can restart. In case of persistent congestion, all the buffers in the data path eventually fill-up and the backpressure signal propagates back to the source(s).

Four backpressure signals are available:

1) from the Out-modules to the fabric output queues;
2) internally to the fabric, from fabric output queues to fabric input queues;
3) from fabric input queues to the In-modules;
4) from the In-module to the input ports.

Backpressure prevents packet losses: however it is not selective, i.e. it blocks all flows, even those which are not responsible for congestion. In [4] we illustrated the benefits achieved by controlling individually unicast flows with centralized arbitration. The same result cannot be easily obtained for multicast, because the number of possible flows traversing the switch grows exponentially (rather than quadratically) with the number of ports N. This implies that switch resource can be hardly assigned per-flow. In particular, both on the ingress and egress side of line-cards packets are stored in a single FIFO queue, regardless of their fanout set.

III. Performance results

We show performance results based on simulation runs exploiting a proprietary simulation environment developed in C language. Statistical significance of the results are assessed by running experiments with an accuracy of 1% under a confidence interval of 95%.

Unless otherwise specified, we refer to a switch with $N = 16$ input and output ports, where all input and output lines run at the same data rate, normalized to 1, and no internal speedup is available. When backpressure is enabled, the high

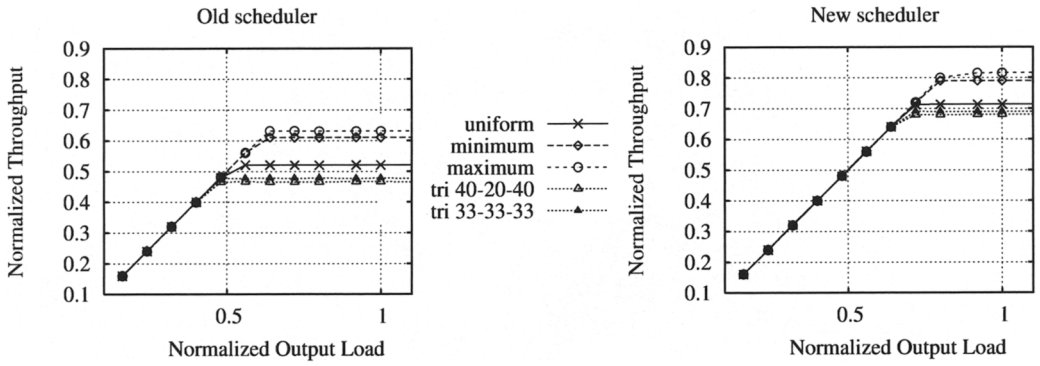

Fig. 2. Performance comparison between the old and the new switching fabric scheduler

threshold, which triggers the backpressure signal, is set to the buffer size, the low threshold is set to 80% of the buffer size.

The average amount of offered traffic at each input (output) is called the input (output) load. Input (output) loads are normalized to line rates: a load equal to 1 means a fully utilized input (output) line. The traffic at the input of a switch is said to be *admissible* if no input load is larger than 1, and no output load is larger than 1.

We first consider Bernoulli arrivals with uniform multicast traffic distribution, both in terms of input/output port distribution and fanout distribution. In other words, all $2^N - 1$ multicast flows, including unicast as a special case, are equally likely. As a consequence, when multicast traffic saturates the inputs, the normalized output load is equal to 8, given the average multicast fanout size of $N/2$. The uniform multicast traffic is admissible only when the input load is smaller than 0.125, for $N = 16$.

We also examine a Bernoulli arrival process in a gathered scenario, where the traffic is gathered over few active input ports ($M = 5$) and equally distributed over all $N = 16$ output ports, with a fanout set chosen according to a non-uniform binomial distribution, with mean fanout $h_m = 3.66$. More precisely, the probability P_f of choosing a fanout set of size f is $P_f = N/h_m \binom{N}{f} (h_m/N)^f (1 - h_m/N)^{N-f}$. This is a traffic pattern well known to be hard to schedule [10]. Indeed, when all inputs are equally loaded, the maximum sustainable traffic leads to a normalized input load which is at most $1/E[f]$, $E[f]$ being the average packet fanout size. If instead the traffic is gathered among few inputs, the normalized input load for sustainable traffic can approach 1, so that the efficiency in serving packets queued at the inputs becomes important on performance. Note that the considered gathered traffic scenario is far from being unrealistic. Multicast applications often generate sustained and long-lasting flows, that may only engage few inputs and several outputs at a given router or switch.

Five packet size distributions are considered:

- constant packet size, all packets of minimum size (mTU, minimum transfer unit of 120 bytes),
- constant packet size, all packets of maximum size (MTU=2000 bytes),
- uniform packet size, ranging from mTU to MTU
- trimodal packet size (120 bytes, 1040bytes, 2000 bytes) with probability 40%, 30%, 40% respectively,
- trimodal packet size (120 bytes, 1040bytes, 2000 bytes) with probabilities 33%, 33%, 33% respectively.

We first show the performance benefit of the newly proposed multicast round robin fabric scheduler; backpressure is activated among all buffering stages. Fig. 2 clearly shows that the new scheduler outperforms the old scheduler. Constant packet size permit to obtain higher throughput mainly thanks to a better efficiency in the use of input fabric FIFO queues. Increasing the packet size variance by using trimodal or uniform distributions worsen performance. This is a peculiar behavior of asynchronous architecture, whereas cell-based synchronous switches suffer less this impairment, thanks to the packet segmentation process at input ports.

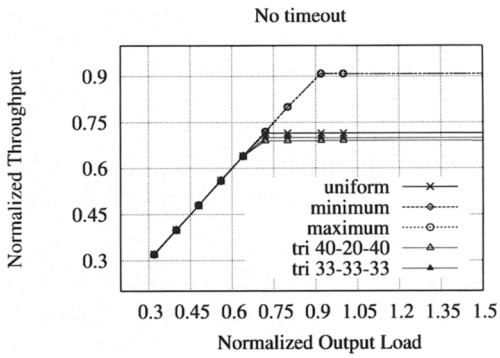

Fig. 3. Performance when no timeout is adopted in the fabric scheduler

Fig. 3 shows that when adopting the new multicast scheduler, the initial timeout used to synchronize grants is not needed. Indeed, whereas performance for trimodal and uniform traffic distributions are not modified, since the performance limitation is due to the inability of filling up input fabric FIFO queues, performance increase significantly, reaching about 90% of link capacity, when using fixed size packets. Indeed, when using fixed size packets, due to the buffering

stage at fabric input, inputs tend to synchronize packet transmissions. For example, when a broadcast packet is scheduled for transmission, all output ports become free at the same time. Being all the packets of the same fixed size, all successive transmissions become exactly synchronized, increasing switch performance. If a timeout expiration is used in the first phase of the multicast scheduling algorithm, this synchronization effect is partially lost, since each multicast packet requires a different number of transmissions to be completely transferred to the outputs belonging to the fanout set. In summary, using a timeout in the scheduling phase either does not provide performance advantages or worsen performance. Thus, no timeout is used in the multicast scheduling phase when adopting the round robin scheduler.

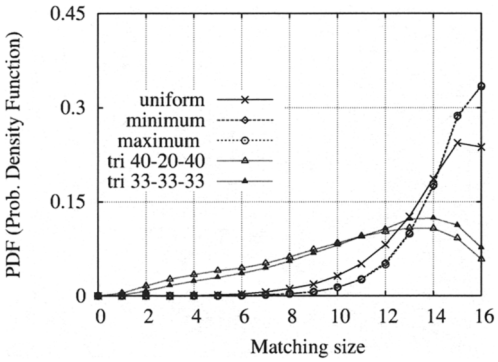

Fig. 4. Matching size distribution in overload under uniform traffic

Again, the asynchronous architecture suffers the increase on the packet size variance due to the independent behavior of schedulers at output ports. This is confirmed by the "matching" size distribution shown in Fig. 4. No matching can be defined in an asynchronous architecture. However, we periodically sample the switching fabric configuration counting the number of active input/output connections: we call this number matching size. The scheduler difficulty in creating large size matching is clearly increasing with increasing variance in the packet size distribution. Thus, the saturation throughput is directly tied to the variance of the packet size distribution.

In Fig. 5, the beneficial effect of backpressure is shown when dealing with non-admissible traffic. Indeed, whereas backpressure activation or deactivation makes no evident difference when the output load is below or close to 1, in deep overload the absence of a backpressure mechanism induces higher losses for trimodal packet size distributions (and marginally higher losses for uniform packet size distribution).

This is a rather counter intuitive behavior. Indeed, when backpressure is active, the packet size distribution in the input FIFO buffers at fabric input is kept constant, regardless of input load, since the source is blocked until 80% of the buffer becomes available. This justifies why no differences are evident when increasing the input load. On the contrary, when backpressure is inactive, in deep overload, small packets have a higher chance of being stored in input FIFOs at fabric inputs. Indeed, when a small packet is transferred, only small packets can be stored in the buffer; when a large packet is transferred, it is enough to store few small packets in the buffer to prevent the possibility of storing a new large packet. Thus, a large number of small packets is stored in fabric buffers, and the average size of packets stored in the input FIFOs decreases as the load increases. This should intuitively lead to an increase in throughput when backpressure is inactive, since the switch should behave similarly to the case of fixed packet size, being the packet size variance decreasing as the input load increases.

However, consider a case when a large packet is transferred from a given input to a set of outputs, and suppose that many small packets are stored in other input queues. Suppose also that small packets are blocked due to contention. This blocking behavior induces a significant throughput decrease, since the transmission time of a large packet with respect to the transmission time of a small packet is significant. In deep overload, this event is more likely to occur with respect to the case of variable packet size distribution, since many small packets are stored in input buffers. Since performance losses are more evident in this "blocking" scenario when inputs store many small packets, this justifies the throughput decrease.

In summary, activating backpressure does not provide performance penalties and helps stabilizing system performance in overload. Thus, we activate the backpressure mechanism in all subsequent simulations. To understand if the penalty provided by the packet size variance is an intrinsic feature of asynchronous architectures or if it depends on the adopted scheduler, we run simulations with all the "weighted" schedulers previously described. Results are reported in Fig. 6 as delays normalized to the packet size, for uniform multicast traffic. We report the results for the LQF scheduler, because no difference were visible by varying the weight metric. The weighted metric does not provide any benefit with respect to the round-robin scheduler; sometimes, performance are even worse, as with trimodal and uniformly variable packet size in both delay and saturation throughput. This results confirms similar observations presented in [10] for synchronous architectures. Performance advantages for more complex weighted schedulers, such as the GS scheduler, are evident only when using more than one FIFO queue at each input, a queue architecture not studied in this paper. Besides, the same general trend by which asynchronous architectures suffer for the packet size variance is maintained.

The same conclusion can be drawn when examining the switch under gathered traffic, in Fig. 7. As expected, in this scenario the maximum achievable throughput is drastically reduced. The general phenomenon of better performance for fixed packet size is even highlighted by this traffic pattern. Results not reported confirm that schedulers based on weighted metrics do not provide evident benefits to switch performance.

IV. Conclusions

Asynchronous architectures suffer variability in packet size distribution, which reduces switch performance both in terms of throughput and delays. This general trend holds regardless

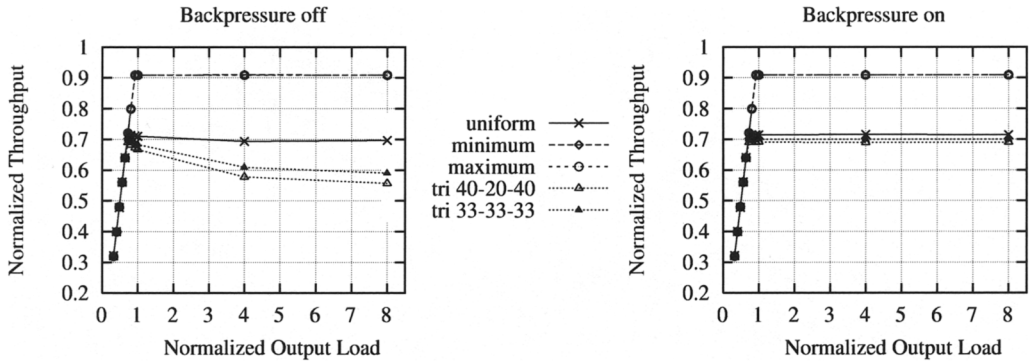

Fig. 5. Backpressure effect

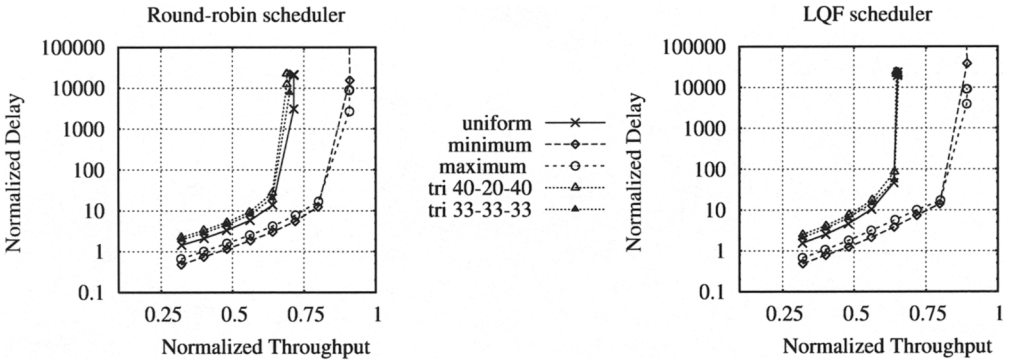

Fig. 6. Packet delays for two different multicast schedulers

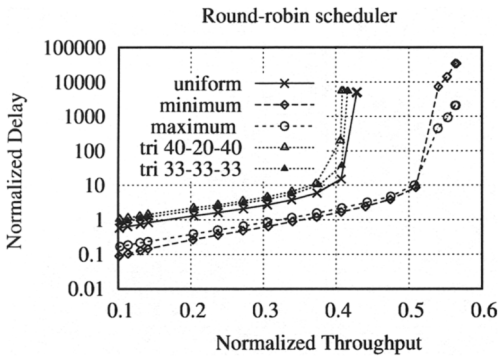

Fig. 7. Packet delays for gathered traffic

of the considered scheduler and the multicast traffic pattern. More complex schedulers based on "weighted" metrics do not provide evident benefits in this scenario, where a single FIFO queue is available. A moderate speedup helps in reducing this performance penalty. Backpressure mechanisms may be beneficial to stabilize performance in overload and do not penalize switch performance in any of the examined scenarios.

REFERENCES

[1] R.W.Kembel and R.Cummings, "Fibre Channel: A Comprehensive Introduction", Northwest Learning Associates, Tucson, AZ, 2000

[2] V.O.K.Li and W.Liao, "Distributed Multimedia Systems", Proc. IEEE, vol. 85, No. 7, pp 1063-1108, July 1997.

[3] S.H. Gary Chan and F.Tobagi, *Distributed Servers Architecture for Networked Video Services*, IEEE/ACM Trans. Networking, Vol. 9, No. 2, pp. 125-136, April 2001.

[4] A.Bianco, P.Giaccone, E.M.Giraudo, F.Neri, E.Schiattarella, "Performance Analysis of Storage Area Network Switches", IEEE Workshop on High Performance Switching and Routing (HPSR 2005), Hong Kong, May 2005.

[5] A.Bianco, P.Giaccone, E.M.Giraudo, F.Neri, E.Schiattarella, "Multicast Support for a Storage Area Network Switch", IEEE Global Telecommunications Conference (GLOBECOM 2006), San Francisco, November 2006.

[6] M.Katevis, G.Passas, D.Simos, I.Papaefstathiou, N.Chrysos, "Variable packet size buffered crossbar (CICQ) switches", IEEE International Conference on Communications (ICC 2004), Paris, France, June 2004.

[7] M.Karol, M.Hluchyj, S.Morgan, "Input versus output queueing on a space division switch", *IEEE Trans. Commun.*,

[8] N.McKeown, A.Mekkittikul, V.Ananthaam, J.Walrand, "Achieving 100% Throughput in an Input-Queued Switch", *IEEE Transcations on Communications*, vol. 47, no. 8, pp. 1260-1267, Aug. 1999.

[9] B.Prabhakar, N.McKeown, R.Ahuja, "Multicast scheduling for input-queued switches", *IEEE J. Sel. Areas Commun.*, vol. 15, no. 5, pp. 855-866, June 1997.

[10] A.Bianco, P.Giaccone, E.Leonardi, F.Neri, C.Piglione "On the Number of Input Queues to Efficiently Support Multicast Traffic in Input Queued Switches", HPSR'03, Torino, Italy, June 2003

[11] L.Mhamdi and M.Hamdi, *Scheduling Multicast Traffic in Internally Buffered Crossbar Switches*, IEEE International Conference on Communications, ICC'04. Paris, June 2004.